Traffic Sign Recognition

LIBO /11/3/2018

Write up for the Project 3: Build a Traffic Sign Recognition Classifier.

Build a Traffic Sign Recognition Project

The goals / steps of this project are the following:

- Load the data set (see below for links to the project data set)
- Explore, summarize and visualize the data set
- Design, train and test a model architecture
- Use the model to make predictions on new images
- Analyze the softmax probabilities of the new images
- Summarize the results with a written report

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. Submission Format.

This is the writeup report of the project completion, along with the other submission files as ipython notebook `Traffic_Sign_Classifier.ipynb` for the source code and the HTML version `Traffic_Sign_Classifier.html` for the output of the source code. Here is a link to my project code.

Data Set Summary & Exploration

1. A basic summary of the data set.

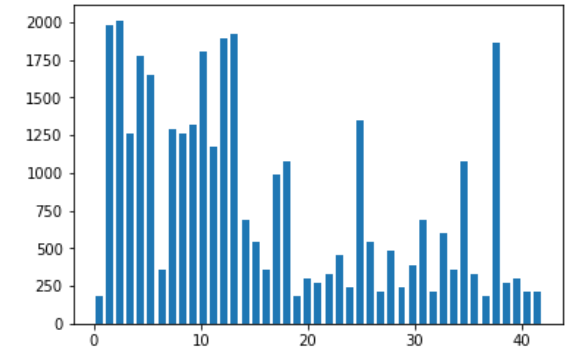
The training, validation and test sets are pre-pickled as `train.p`, `valid.p` and `test.p` respectively. I use pickle.load() file to load thoses pickle files as dictinaries for subtracting suitable data for use in the following precedures.

- The size of training set is 34799,
- The size of the validation set is 4410,
- The size of test set is 12630,
- The shape of a traffic sign image is (32, 32, 3),
- The number of unique classes/labels in the data set is 43.

2. Include an exploratory visualization of the dataset.

Here is an exploratory visualization of the data set. It is a bar chart showing how the data is distributed for 43 classes.

It seems that the samples for each class is not uniformly distributed, some classes might be lack of enough samples (the 0 class contains only 180 samples for example), and the result would be more likely biased to the classes with more samples, while loss of enough weights to the classes with few samples.



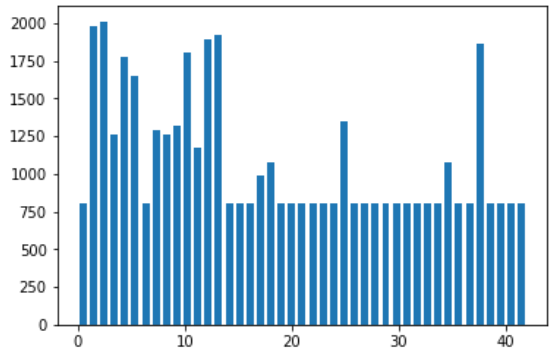
To solve this problem, some data augmentation techniques are applied as the random translation, random_scale, random_warp, and random brightness.

The code for the data augmentation are listed as follows:

```
def random_translate(img):
    rows,cols,_ = img.shape

    # allow translation up to px pixels in x and y directions
    px = 2
    dx,dy = np.random.randint(-px,px,2)

    M = np.float32([[1,0,dx],[0,1,dy]])
    dst = cv2.warpAffine(img,M,(cols,rows))

    dst = dst[:,:,np.newaxis]

    return dst

def random_scaling(img):
    rows,cols,_ = img.shape

    # transform limits
    px = np.random.randint(-2,2)

    # ending locations
    pts1 = np.float32([[px,px],[rows-px,px],[px,cols-px],[rows-px,cols-px]])

    # starting locations (4 corners)
    pts2 = np.float32([[0,0],[rows,0],[0,cols],[rows,cols]])

    M = cv2.getPerspectiveTransform(pts1,pts2)

    dst = cv2.warpPerspective(img,M,(rows,cols))

    dst = dst[:,:,np.newaxis]

    return dst

def random_warp(img):

    rows,cols,_ = img.shape

    # random scaling coefficients
    rndx = np.random.rand(3) - 0.5
```

```
    rndx *= cols * 0.06   # this coefficient determines the degree of warping
    rndy = np.random.rand(3) - 0.5
    rndy *= rows * 0.06

    # 3 starting points for transform, 1/4 way from edges
    x1 = cols/4
    x2 = 3*cols/4
    y1 = rows/4
    y2 = 3*rows/4

    pts1 = np.float32([[y1,x1],
                       [y2,x1],
                       [y1,x2]])
    pts2 = np.float32([[y1+rndy[0],x1+rndx[0]],
                       [y2+rndy[1],x1+rndx[1]],
                       [y1+rndy[2],x2+rndx[2]]])

    M = cv2.getAffineTransform(pts1,pts2)

    dst = cv2.warpAffine(img,M,(cols,rows))

    dst = dst[:,:,np.newaxis]

    return dst

def random_brightness(img):
    shifted = img + 1.0   # shift to (0,2) range
    img_max_value = max(shifted.flatten())
    max_coef = 2.0/img_max_value
    min_coef = max_coef - 0.1
    coef = np.random.uniform(min_coef, max_coef)
    dst = shifted * coef - 1.0
    return dst
```

After the data augmentation, the class distribution might be more balanced as the following bar chart shows:



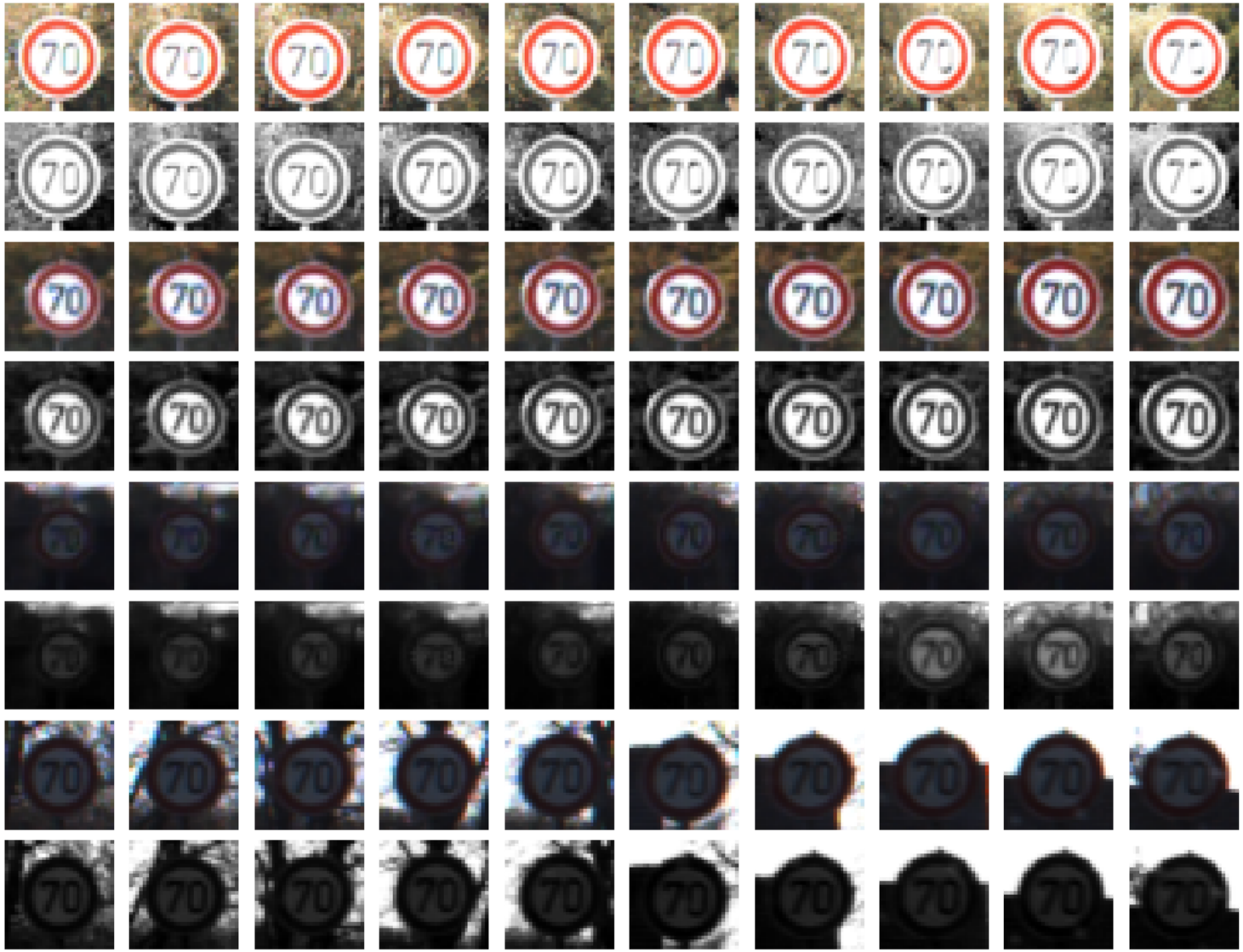
Design and Test a Model Architecture

1. Describe how you preprocessed the image data. What techniques were chosen and why did you choose these techniques? Consider including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization, etc. (OPTIONAL: As described in the "Stand Out Suggestions" part of the

rubric, if you generated additional data for training, describe why you decided to generate additional data, how you generated the data, and provide example images of the additional data. Then describe the characteristics of the augmented training set like number of images in the set, number of images for each class, etc.)

As a first step, I decided to convert the images to grayscale for two reasons, one for simplicity and the other for data reduction. The gray scale images only contain one image plane (containing the gray scale intensity values), and will only need to process 1/3 of the data compared to the color image. However, it is worthy to notice that the drawbacks of grayscale are also evident that the quantity of the data is reduced and the potential risk of losing information should be noticed. This traffic sign classification is not so relevant to the color information, so it is quite worth applying grayscale tech for reducing training time in CNN networks.

Here is an example of comparison of traffic sign image before and after grayscaling.



As a last step, I normalized the image data so that they are of approximately the same scale, so that the learning rate would cause corrections in each directions evenly. Otherwise, we might over compensating a correction in one weight dimension while undercompensating another.

2. Describe what your final model architecture looks like including model type, layers, layer sizes, connectivity, etc.) Consider including a diagram and/or table describing the final model.

My final model is a modified LeNet framework, which consisted of the following layers:

Layer	Description
Input	32x32x1 gray_scaled image
Convolution 5x5	1x1 stride, valid padding, outputs 28x28x6
ELU	Activation function
Dropout	drop out 40% nodes
Max pooling	2x2 stride, outputs 14x14x6
Convolution 5x5	1x1 stride, valid padding, outputs 10x10x16
ELU	Activation function
Dropout	drop out 30% nodes
Max pooling	2x2 stride, outputs 5x5x6
Flatten	outputs 400
Fully connected	outputs 120
ELU	Activation function
Fully connected	outputs 84
ELU	Activation function
Fully connected	outputs 43

The code for the modified LeNet framework is following:

```
def LeNet(x):
    # Arguments used for tf.truncated_normal, randomly defines variables for the weights and biases for each layer
    mu = 0
    sigma = 0.1

    # SOLUTION: Layer 1: Convolutional. Input = 32x32x1. Output = 28x28x6.
    conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 1, 6), mean = mu, stddev = sigma))
    conv1_b = tf.Variable(tf.zeros(6))
    conv1   = tf.nn.conv2d(x, conv1_W, strides=[1, 1, 1, 1], padding='VALID') + conv1_b

    # SOLUTION: Activation.
    conv1 = tf.nn.elu(conv1, name = 'convolution0')

    # SOLUTION: Dropout
    conv1 = tf.nn.dropout(conv1, keep_prob)

    # SOLUTION: Pooling. Input = 28x28x6. Output = 14x14x6.
    conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID', name='convolution1')

    # SOLUTION: Layer 2: Convolutional. Output = 10x10x16.
    conv2_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 6, 16), mean = mu, stddev = sigma))
    conv2_b = tf.Variable(tf.zeros(16))
    conv2   = tf.nn.conv2d(conv1, conv2_W, strides=[1, 1, 1, 1], padding='VALID') + conv2_b

    # SOLUTION: Activation.
    conv2 = tf.nn.elu(conv2,name='convolution2')
```

```
    # SOLUTION: Dropout
    conv2 = tf.nn.dropout(conv2, keep_prob2)

    # SOLUTION: Pooling. Input = 10x10x16. Output = 5x5x16.
    conv2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID',name='convolution3')

    # SOLUTION: Flatten. Input = 5x5x16. Output = 400.
    fc0   = flatten(conv2)

    # SOLUTION: Layer 3: Fully Connected. Input = 400. Output = 120.
    fc1_W = tf.Variable(tf.truncated_normal(shape=(400, 120), mean = mu, stddev = sigma))
    fc1_b = tf.Variable(tf.zeros(120))
    fc1   = tf.matmul(fc0, fc1_W) + fc1_b

    # SOLUTION: Activation.
    fc1    = tf.nn.elu(fc1)

    # SOLUTION: Layer 4: Fully Connected. Input = 120. Output = 84.
    fc2_W  = tf.Variable(tf.truncated_normal(shape=(120, 84), mean = mu, stddev = sigma))
    fc2_b  = tf.Variable(tf.zeros(84))
    fc2    = tf.matmul(fc1, fc2_W) + fc2_b

    # SOLUTION: Activation.
    fc2    = tf.nn.elu(fc2)

    # SOLUTION: Layer 5: Fully Connected. Input = 84. Output = 43.
    fc3_W  = tf.Variable(tf.truncated_normal(shape=(84, 43), mean = mu, stddev = sigma))
    fc3_b  = tf.Variable(tf.zeros(43))
    logits = tf.matmul(fc2, fc3_W) + fc3_b

    return logits


x = tf.placeholder(tf.float32, (None, 32, 32, 1))
y = tf.placeholder(tf.int32, (None))
one_hot_y = tf.one_hot(y, 43)
keep_prob = tf.placeholder(tf.float32) # probability to keep units
keep_prob2 = tf.placeholder(tf.float32)
```

3. Describe how you trained your model. The discussion can include the type of optimizer, the batch size, number of epochs and any hyperparameters such as learning rate.

To train the model, I used 70 epochs and 128 samples for each batch, the learning rate is set to 0.001. Since this is a classification problem of predicting 43 classes which is quite similar to predicting 10 classes in LeNet MNIST example, so we can emulate what Professor Yann LeCun does, with AdamOptimizer to optimize the loss function described by cross entropy.

4. Describe the approach taken for finding a solution and getting the validation set accuracy to be at least 0.93. Include in the discussion the results on the training, validation and test sets and where in the code these were calculated. Your approach may have been an iterative process, in which case, outline the steps you took to get to the final solution and why you chose those steps. Perhaps your solution involved an already well known implementation or architecture. In this case, discuss why you think the architecture is suitable for the current problem.

My final model results were:

- training set accuracy of 99.5%
- validation set accuracy of 98.0%
- test set accuracy of 91.2%

The well known LeNet architecture is chosen for this project, only a slight modification of the output.

The LeNet arichitecture is originally designed for classfiying the digits of the class from 1 to 10. LeNet is a type of CNN network useful not only for classifying the 10 digits in MNIST dataset. Our project `Traffic Sign Classifier Project` is also a classification project on classifying 43 different traffic signs which is quite similar to what LeNet does.

The final model's accuracy on training is 99.5%, but this result is not so important as the validation accuracy 98.0%,since the validation data is thrown in the CNN framwork to test the state-of-the-art accuracy after updating the weights at each epoch. But the most import index for the final model's accuracy might be due to the test accuracy. The test dataset never participate in the training process, and keep independent from the training set and the validation set. So the test accuracy evaluated on the stand-alone test set might be more convincing to the customers to judge the effectiveness of the LeNet framework.

It is worth noticing from the past experience that the problem of such Deep Neural Networks(DNN) is overfitting. The model learns to classify only the training examples instead of learning decision boundaries capable of classifying generic instances. The drop out seems to be very useful. Before the drop out applies, the modified LeNet mightbe overfitted respect to the training sets, but when the trained framework is tested on some extreme cases not met in the training sets, the prediction might not be so robust. For example, the last picture under occluded or some complicated background cluttered behind, the prediction of the network without dropout might give a wrong answer. It is commonly accepted that the dropout could work well in practice because it prevents the co-adaption of neurons during the training phase.

With some proper drop out preventing the co-adaptations which might not generalized to unseen data, the dropped out neurons could be regarded as some kind of hidden units unreliable. Therefore, the hidden units cannot be relied on other specific units to correct its mistakes.

The other modification of the LeNet framework is to replace the RELU with ELU(Exponential Linear Units) for the activation function. The original RELU might be always output the same value if some large negative bias term is met in some particular training process. Once the RELU is "dead", it is unlikely to recover, because the gradient at 0 is also 0. The ELU gives a solution on providing some small positive gradient for negative inputs in attempt to give a chance to recover. The switch from RELU to ELU brings about the lift of accuracy in predicting the last example picture from 57% to 97%.

Test a Model on New Images

1. Choose five German traffic signs found on the web and provide them in the report. For each image, discuss what quality or qualities might be difficult to classify.

Here are five German traffic signs that I found on the web:

These 5 pictures are not difficult to classifier and my network shows that all of the pictures are correctly classified. The 5 traffic signs are chosen under different illuminated condition. The accuracy might be affected by the dark background, but the network give the right prediction even the traffic sign is captured at night.

2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. At a minimum, discuss what the predictions were, the accuracy on these new predictions, and compare the accuracy to the accuracy on the test set (OPTIONAL: Discuss the results in more detail as described in the "Stand Out Suggestions" part of the rubric).

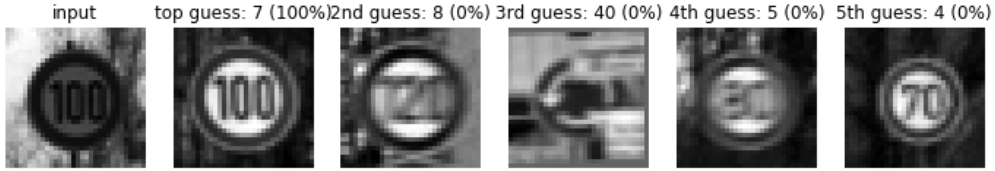
Here are the results of the prediction:

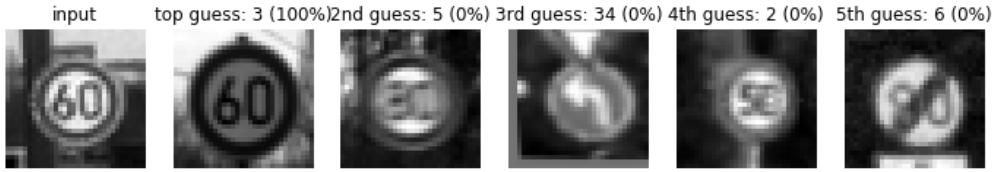
Image	Prediction
End of all speed and passing limits	End of all speed and passing limits
Slippery road	Slippery road
Pedestrians	Pedestrians
No entry	No entry
No passing for vehicles over 3.5 metric tons	No passing for vehicles over 3.5 metric tons

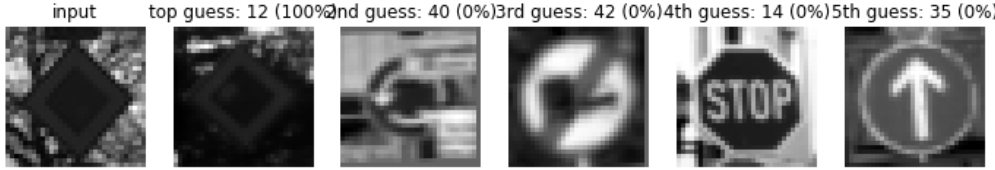
The model was able to correctly guess 5 of the 5 traffic signs, which gives an accuracy of 100%. This compares favorably to the accuracy on the test set of 91.2%.

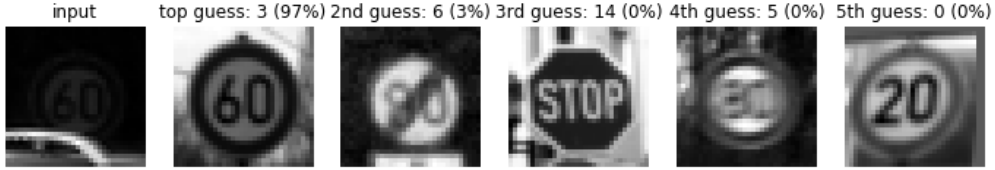
3. Describe how certain the model is when predicting on each of the five new images by looking at the softmax probabilities for each prediction. Provide the top 5 softmax probabilities for each image along with the sign type of each probability. (OPTIONAL: as described in the "Stand Out Suggestions" part of the rubric, visualizations can also be provided such as bar charts)

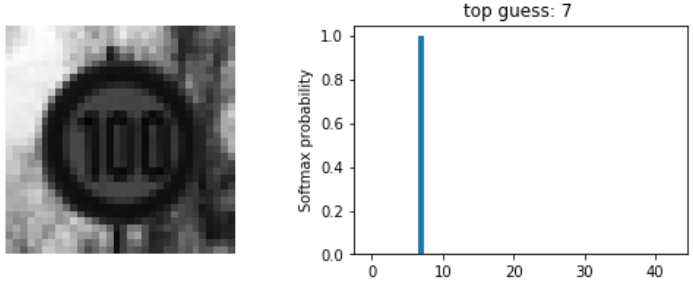
The code for making predictions on my final model is located in the 41st and 42nd cells of the Jupyter notebook. 10 samples of 32x32 images are chosen randomly from the German traffic sign dataset and the input pictures are located at the first column, while the top 3 predictions based on the softmax are located on the remaining 3 columns resectively.

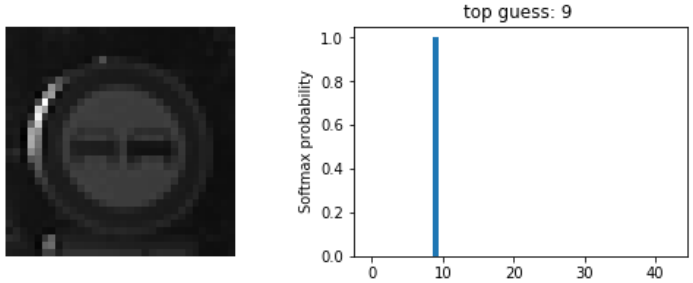


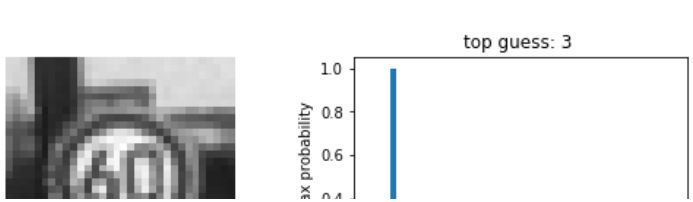


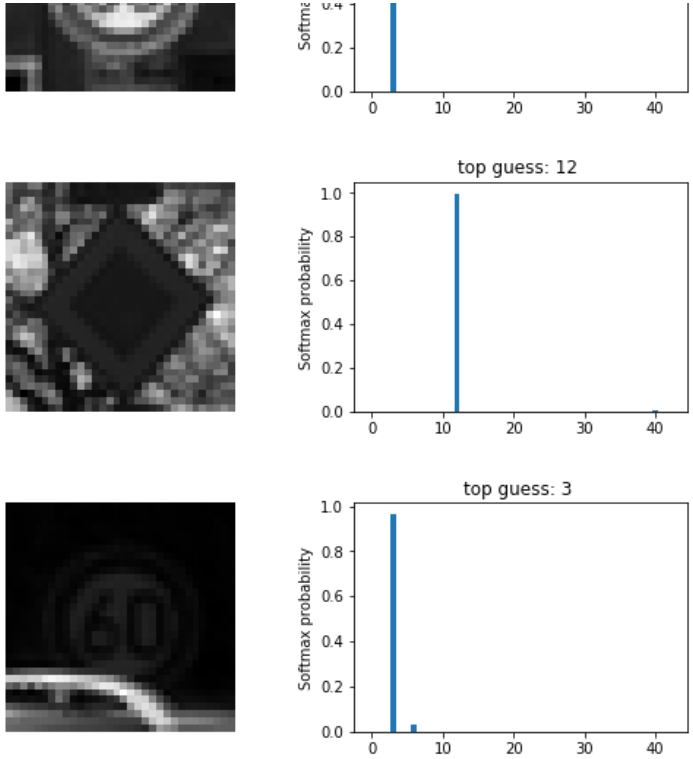

input
top guess: 12 (100%)
2nd guess: 40 (0%)
3rd guess: 42 (0%)
4th guess: 14 (0%)
5th guess: 35 (0%)
STOP


input
top guess: 3 (97%)
2nd guess: 6 (3%)
3rd guess: 14 (0%)
4th guess: 5 (0%)
5th guess: 0 (0%)
60
STOP
20


top guess: 7
Softmax probability
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40


top guess: 9
Softmax probability
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40


top guess: 3
1.0
0.8
0.6



For all the 5 images, the model is correct at their top guess, however, however the guess is not always so confident.

For the first image, which shows Speed limit (100km/h), the top five soft max probabilities of this image are presented as

Probability	Prediction
1.00	Speed limit (100km/h)
.00	Speed limit (120km/h)
.00	Roundabout mandatory
.00	Speed limit (80km/h)
.00	Speed limit (70km/h)

Since the top 1 prediction is the same as the actual traffic sign Class Id, the first prediction is correct.

For the second image which shows No passing, the top five soft max probabilities of this image are presented as

Probability	Prediction
1.00	No passing
.00	No passing for vehicles over 3.5 metric tons
.00	Slippery road
.00	Dangerous curve to the left
.00	Ahead only

Since the top 1 prediction is the same as the actual traffic sign Class Id, the prediction of the second image is correct.

For the third image which shows Speed limit (60km/h), the top five soft max probabilities of this image are presented as

Probability	Prediction
1.00	Speed limit (60km/h)
.00	Speed limit (80km/h)
.00	Turn left ahead
.00	Speed limit (50km/h)
.00	End of speed limit (80km/h)

Since the top 1 prediction is the same as the actual traffic sign Class Id, the prediction of the third image is correct.

For the fourth image which shows Priority road, the top five soft max probabilities of this image are presented as

Probability	Prediction
1.00	Priority road
.00	Roundabout mandatory
.00	End of no passing by vehicles over 3.5 metric tons
.00	Stop
.00	Ahead only

Since the top 1 prediction is the same as the actual traffic sign Class Id, the prediction of the fourth image is correct.

For the last image which shows Speed limit (60km/h), the top five soft max probabilities of this image are presented as

Probability	Prediction
0.97	Speed limit (60km/h)
.03	End of speed limit (80km/h)
.00	Stop
.00	Speed limit (80km/h)
.00	Speed limit (20km/h)

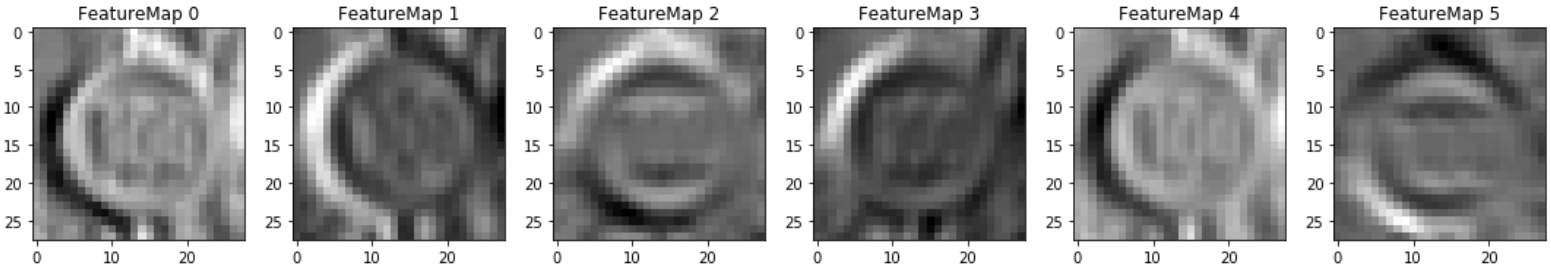
Since the top 1 prediction is the same as the actual traffic sign Class Id, the prediction of the last image is correct. The extremely dark background of the 60km/h speed limit sign might be a little difficult for the Lenet training framwork, so the top 1 guess is not 100%, only 97% sure of the right answer.

(Optional) Visualizing the Neural Network (See Step 4 of the Ipython notebook for more details)

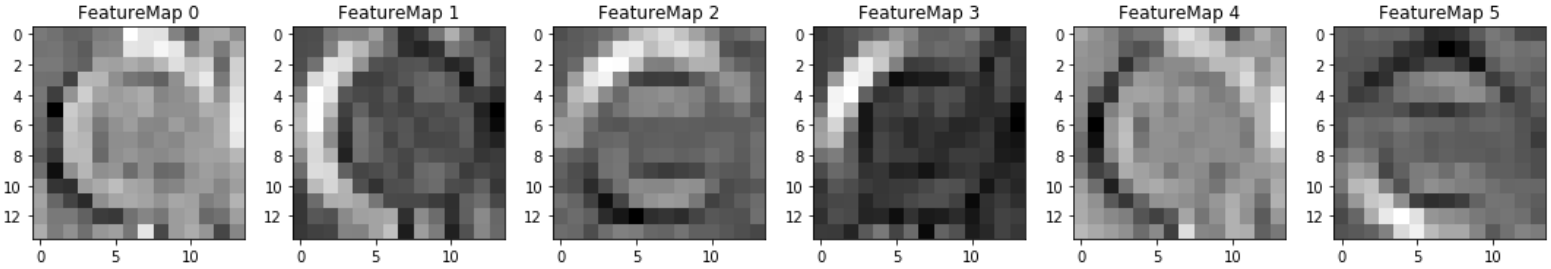
1. Discuss the visual output of your trained network's feature maps. What characteristics did the neural network use to make classifications?

Taking the first test image `Speed limit (100km/h)` as an example, we observe the following feature map presentations by using the outputFeatureMap() function.

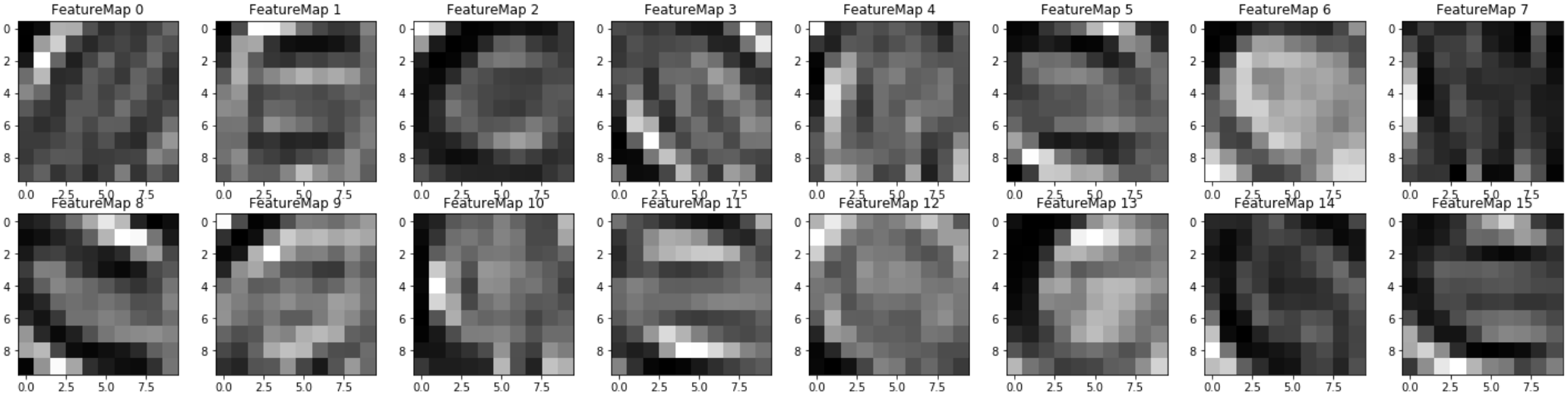
The visual output of the trained network is presented as follows: The first convolution layer of 5x5 kernel size with relu activation might be represented as the following 6 images:



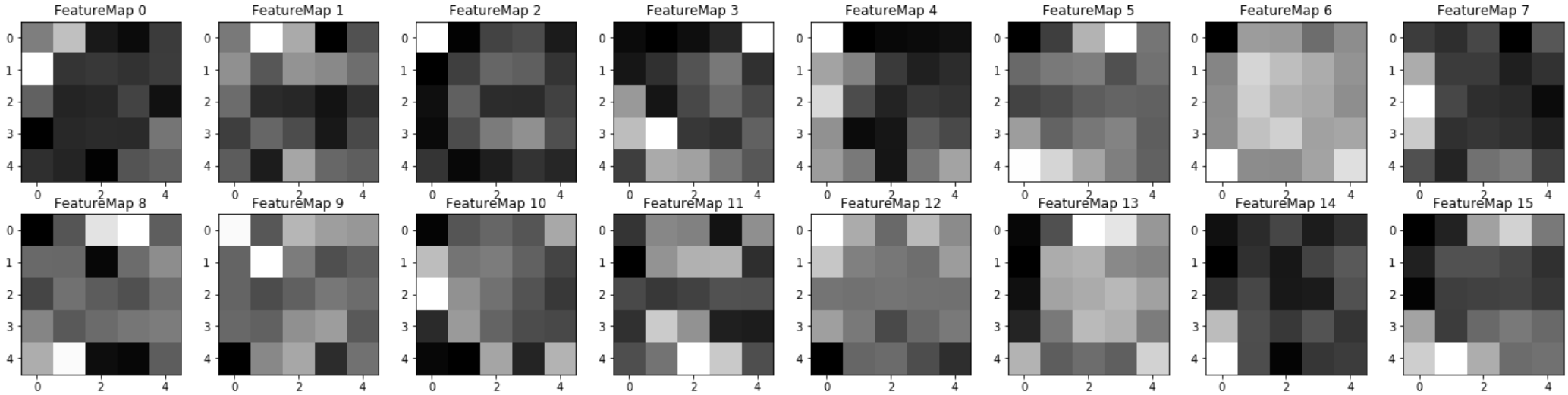
After the first maxpooling, the output images would be the follwing 6 images



The second convolution layer after the relu activation can be visualized by feature map arrayed as the following 16 images:



After the second maxpooling, the feature_map output is visualized as the 16 images:


FeatureMap 0
FeatureMap 1
FeatureMap 2
FeatureMap 3
FeatureMap 4
FeatureMap 5
FeatureMap 6
FeatureMap 7
FeatureMap 8
FeatureMap 9
FeatureMap 10
FeatureMap 11
FeatureMap 12
FeatureMap 13
FeatureMap 14
FeatureMap 15